Push relabel maximum flow algorithm

Seminarski rad u okviru kursa
Konstrukcija i analiza algoritama 2
Matematički fakultet

Anđela Damnjanović
mi19059@alas.matf.bg.ac.rs

5. maj 2024.

Sažetak

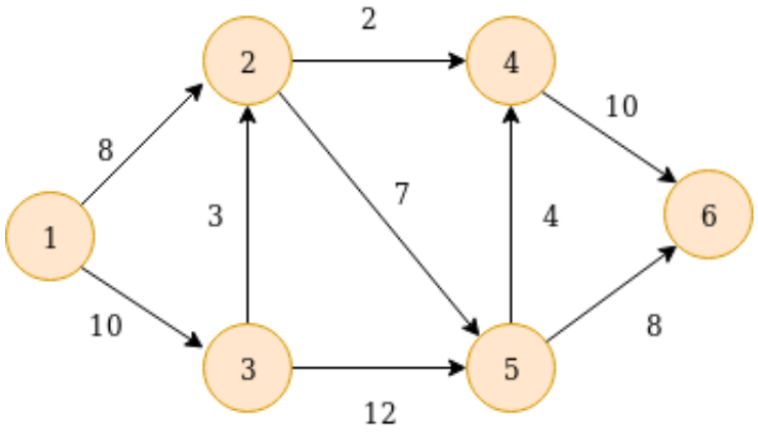
Teorija grafova i grafovski problemi imaju veoma važno mesto u matematičkoj teoriji, ali ujedno nalaze i brojne praktične primene sa kojima se ljudski rod svakodnevno susreće — od navigacije pa sve do generisanja rasporeda časova ili modela atoma i molekula. Upravo zbog njihove velike rasprostranjenosti, razvijeni su mnogi algoritmi koji imaju za cilj da što efikasnije reše pomenute probleme. U skladu sa tim, centralna tema ovog rada je upravo opis jednog veoma bitnog algoritma koji služi za optimizaciju transportne mreže, a koji je poznat pod nazivom Push-relabel maximum flow algorithm.

Sadržaj

1 Uvod

Grafovi su matematičke strukture koje se sastoje od skupa *čvorova* (koji se, po konvenciji, obeležava slovom V) i skupa *grana* (koji se obeležava slovom E). Oni predstavljaju pogodnu strukturu podataka za modelovanje najrazličitijih problema na koje ljudi nailaze. Mogu se koristiti za pravljenje modela molekula u hemiji, za modelovanje povezanosti korisnika na društvenim mrežama, za vizualizaciju povezanosti mesta na geografskoj karti i tako dalje. Stoga ni ne treba da čudi što su svoju primenu našli i u računarstvu — grafovi imaju veoma širok spektar primena poput klasterovanja, istraživanja podataka, segmentacije slika [4]. Naravno, ovo su samo neke oblasti računarstva gde su grafovi korisni.

Za sve probleme koji se mogu predstaviti grafovima, definisani su odgovarajući algoritmi koji ih rešavaju. Međutim, ne mogu se svi algoritmi primeniti na sve grafove. Na primer, nalaženje minimalnog razapinjućeg stabla nije moguće primeniti na usmerene grafove, a nalaženje Ojlerovog puta neće biti moguće ako je broj čvorova sa neparnim stepenom različit od 0 i 2 [2]. Analogno, ni push-relabel maximum flow algoritam, koji je centralna tema ovog rada, ne može se primeniti na svaki graf. Da bi pomenuti algoritam bio primenljiv, graf mora biti *transportna mreža*, tj. neophodno je da graf bude povezan, usmeren, težinski i sadrži tačno jedan čvor sa ulaznim stepenom 0 iz koga će se propuštati tok (stoga prikladno nazvan *izvor*) i jedan čvor sa izlaznim stepenom 0 u koji će sav tok uvirati (te je nazvan *ponor*)[1]. Takođe, neophodno je i da grane grafa budu pozitivne težine. Svaka ovakva težina označava najveću količinu toka koju je moguće propustiti kroz granu i naziva se *kapacitet* grane. Primer jedne transportne mreže prikazan je na slici 1.



Slika 1: Grafički prikaz transportne mreže [3]

[1]Ukoliko u grafu postoji više izvora i ponora, potrebno je dodati nova dva čvora: jedan od koga će grane voditi ka svakom izvoru i jedan do koga će grane voditi od svakog ponora, čime se problem svodi na već poznati problem.

Osobine toka koje je potrebno zadovoljiti su:

1. Količina toka koja se propušta kroz granu ne sme biti manja od 0 i ne sme prevazilaziti kapacitet te grane
2. Ukupna količina toka koja uđe u čvor mora biti jednaka količini toka koja taj isti čvor napušta [2].

Kao što se iz naziva može zaključiti, transportne mreže su pogodne za modelovanje fizičkog transporta sa jednog mesta na drugo, npr. mogu se koristiti za modelovanje cevovoda ili pak saobraćaja. Međutim, transportne mreže mogu naći svoju primenu i za modelovanje rasporeda sedenja ili rasporeda časova, za višeciljne optimizacije, kao i za segmentaciju slika [5, 3].

U nastavku će biti opisani dosadašnji rezultati poznatih matematičara u sekciji 2, biće izveden odgovarajući algoritam u sekciji 3 koji će biti testiran na različitim ulazima u delu 4, te će se na osnovu njih donostiti zaključak u sekciji 5.

2 Opis problema i dosadašnji rezultati

Nakon upoznavanja sa osnovnim pojmovima koji su potrebni za razumevanje algoritma, moguće je posvetiti pažnju detaljnom opisu problema. Neka je data transportna mreža. Potrebno je pronaći maksimalni tok kroz nju, tj. potrebno je odrediti koja je najveća količina supstance koja se može pustiti iz izvora i, putujući granama zadatog grafa prema pravilima opisanim u 1, stići do ponora.

Jedan od prvih algoritama koji rešavaju dati problem osmišljen je davne 1956.godine i nosi naziv Ford[3]-Fulkersonov[4] algoritam. On pripada grupi pohlepnih algoritama i zasniva se na *povećavajućim putevima*. Sam algoritam je veoma jednostavan: potrebno je najpre pronaći put od izvora do ponora i pamtiti koji je najmanji kapacitet grana koje se nalaze na tom putu. Zatim se od izvora ka ponoru propušta odgovarajuća količina supstance (jednaka upravo pronađenom minimalnom kapacitetu) i dati postupak se ponavlja dokle god postoji put između izvora i ponora.

Dakle, algoritam se završava onda kada više nema uvećavajućih puteva. Međutim, ne postoji garancija da će se algoritam uopšte završiti, pa on samim tim garantuje ispravan rezultat samo u slučaju kada se algoritam može završiti u konačno mnogo koraka. Složenost datog algoritma je O(Ef) gde je f vrednost maksimalnog toka kroz graf, jer je za pronalaženje puta od izvora do ponora potrebno obići sve grane grafa, dok je broj uvećavajućih puteva koje treba pronaći između 1 i f.

Kako bi se rešio problem terminacije Ford-Fulkersonovog algoritma, autori Edmonds[5] i Karp[6] su nezavisno došli do načina da praktično implementiraju Ford-Fulkersonov algoritam tako da se on stalno zaustavlja, te je ova implementacija nazvana po njima. Jedina izmena u odnosu na originalni algoritam leži u tome što se za uvećavajući put bira *najkraći* takav put u grafu. Složenost datog algoritma je O(E^2V). Ponovo, vreme

[2] važi za sve čvorove osim za izvor i ponor

[3] Lester Rendolf Ford (1927-2017) američki matematičar koji se specijalizovao upravo za probleme transportnih mreža

[4] Delbert Rej Fulkerson (1924-1976) američki matematičar

[5] Džek Edmonds (1934-), američki informatičar i matematičar, najpoznatiji po svom doprinosu u poljima optimizacije, kombinatorike i diskretne matematike

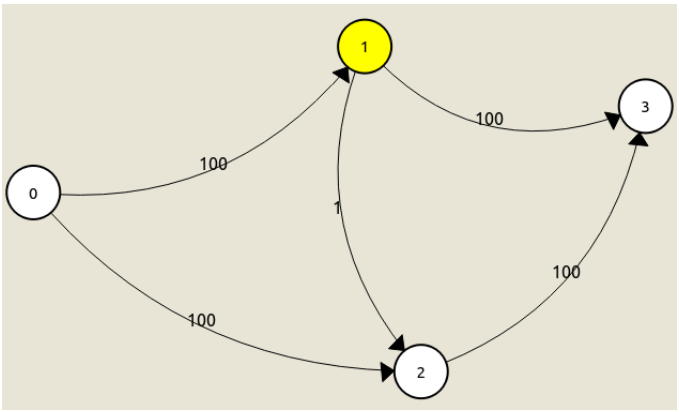
[6] Ričard Maning Karp (1935-), američki informatičar najpoznatiji po doprinosima u oblasti teorije algoritama, dobitnik Tjuringove nagrade

da se nađe uvećavajući put zahteva O(E) vremena, dok je najveći broj povećavanja puteva O(EV).

Kao što se može videti, pomenuti algoritmi nisu toliko efikasni ili čak nisu uvek terminirajući. Stoga je razvijen još jedan algoritam, koji je i efikasniji od Edmonds-Karpovog, ali se i uvek garantuje njegovo zaustavljanje za razliku od generičnog Ford-Fulkersonovog algoritma. Taj algoritam dobio je naziv Push-relabel maximum flow algoritam i zasniva se na propuštanju određene količine toka iz jednog čvora u njemu susedan (ova operacija, koja se čak našla i u samom imenu algoritma, nazvana je push) ukoliko je to moguće i u ponovnom obeležavanju čvorova (operacija relabel) ukoliko to nije moguće. Jedna od osnovnih razlika u odnosu na prethodne pristupe podrazumeva postojanje takozvanog *predtoka* (eng. *preflow*) koji zasićuje sve grane koje izlaze iz izvora. Još jedna novina u odnosu na već postojeće algoritme jeste obeležavanje čvorova. Ideja je da se tok može propustiti iz jednog čvora u drugi pod uslovom da se prvi nalazi na većoj "visini"od drugog. Stoga je za svaki čvor bitno čuvati njegovu trenutnu visinu [1].

3 Implementacija

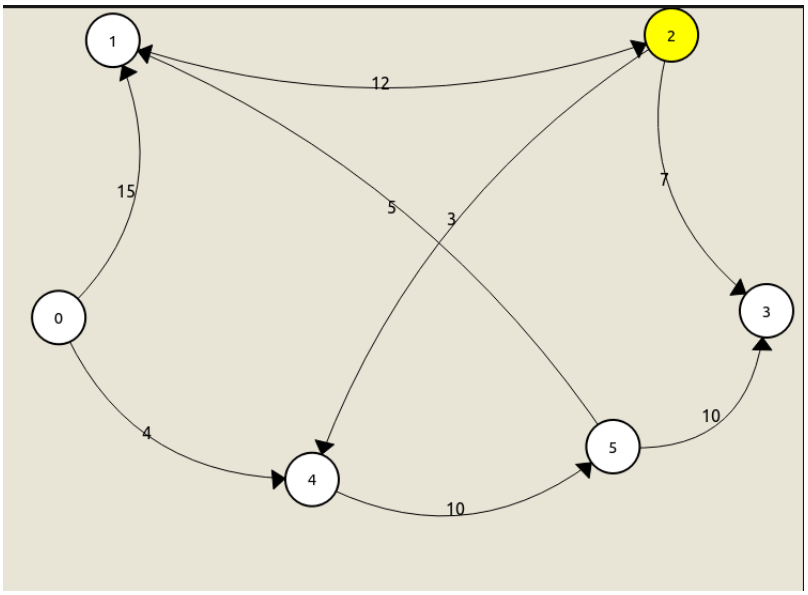
Za implementaciju pomenutog algoritma korišćeni su programski jezik C++ i QtCreator. Primer vizualizacije jedne transportne mreže može se naći na slikama 2, 3 i 4. Čvorovi i grane grafa, kao i sam graf predstavljeni opisani su odgovarajućim klasama. Od značajnih atributa klase Node izdvajaju se polje *excessFlow* koje govori koliko "viška"toka se nalazi u datom čvoru i polje *height* koje govori koja je visina datog čvora. Sa druge strane, klasi Edge dodat je atribut *flow* koji govori koja količina toka je propuštena kroz datu granu [7].



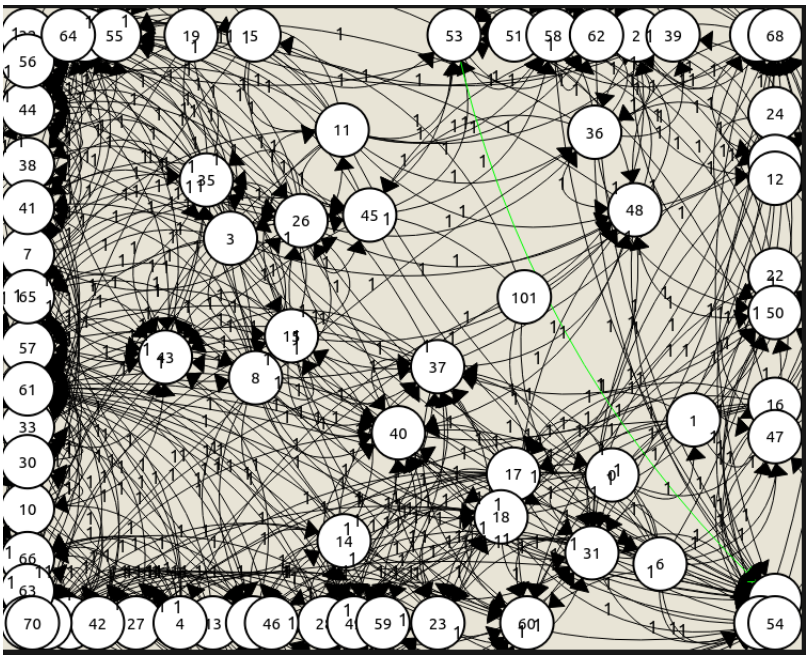
Slika 2: Prikaz transportne mreže sa 4 čvora

Najpre je potrebno napraviti odgovarajući rezidualni graf koji će sadržati sve grane originalnog grafa, ali i *rezidualne* grane granama koje vode od izvora ka njegovim susedima. Zatim je potrebno napraviti odgovarajući *predtok*, tj. od izvora do njegovih suseda propustiti maksimalni mogući

[7] polje koje čuva težinu grane (kapacitet u slučaju push-relabel algoritma) već postoji u implementaciji



Slika 3: Prikaz transportne mreže sa 6 čvorova



Slika 4: Prikaz transportne mreže sa 80 čvorova

tok (to je količina toka koja odgovara kapacitetu grane). Istovremeno, vrednosti *excessFlow* promenljive kod čvorova koji su susedni izvoru postavljaju se na vrednosti kapaciteta grane koja vodi od izvora do njih, dok

Algorithm 1 Generic push-relabel maximum flow algorithm

Require: graph with one source node, one sink node and edges with positive capacities

```
    Initialization:
        makeResidualGraph
        initializePreflow
while ∃ anactivenode do
    while excessFlow > 0 do
        if can push from active node then
            push flow to a neighbouring node
        else if !can push then
            relabel active node
        end if
    end while
        remove node from the list of active nodes
end while
```

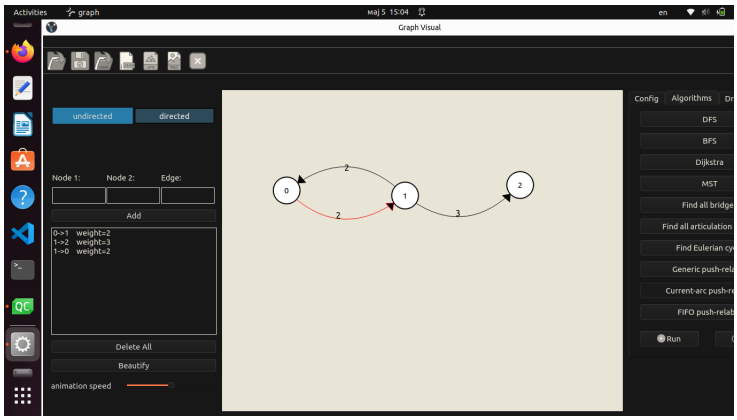
se vrednost iste promenljive u izvoru smanjuje za odgovarajuću vrednost. Nakon što je tok propušten kroz granu originalnog grafa, potrebno je vrednost toka odgovarajuće rezidualne grane umanjiti za vrednost kapaciteta grane. Ovim je završen proces inicijalizacije.

Svi čvorovi koji imaju višak toka u sebi moraju ga proslediti do ponora (ili deo vratiti nazad u izvor ukoliko ne postoji način da se sav tok propusti do ponora). Dakle, unutrašnji čvorovi grafa ne smeju u sebi imati viška toka kada se algoritam završi. Zato se oni proglašavaju *aktivnim* čvorovima, tj. čvorovima koji mogu da propuste tok. Nakon inicijalnog koraka jedini aktivni čvorovi su neposredni susedi izvornog čvora, što se može promeniti tokom izvršavanja algoritma. Zato je potrebno obići čvorove u svakoj iteraciji.

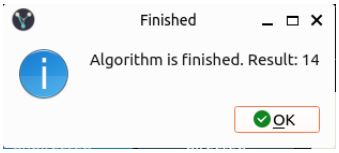
Ako postoji aktivan čvor, potrebno ga je obraditi. Ukoliko je visina nađenog čvora veća od visine nekog od njegovih suseda i ako je tok kroz granu koja ih povezuje manji od kapaciteta te grane, moguće je propustiti tok iz aktivnog čvora u njegovog suseda. Količina toka koja će biti propuštena je *delta = min(u->excessFlow, e->capacity - e->flow)*, gde je u trenutni aktivan čvor, a e grana koja vodi od u do njegovog suseda. Nakon propuštanja toka iz aktivnog čvora, potrebno je napraviti odgovarajuću rezidualnu granu koja će imati isti kapacitet kao i grana originalnog grafa. Zatim se povećava vrednost toka grane kroz koju je propušten tok, dok se vrednost toka napravljene rezidualne grane smanjuje za istu vrednost.

Sa druge strane, ukoliko nije moguće propustiti tok kroz aktivni čvor, potrebno je dati čvor ponovo obeležiti. Obeležavanje podrazumeva promenu visine aktivnog čvora. Nova visina čvora određuje se tako što se nađe minimum visinâ svih njegovih suseda i toj vrednosti se doda 1. Dalje algoritam teče na isti način sve dok svi čvorovi osim izvora i ponora imaju višak toka u sebi. Vrednost maksimalnog toka može se pročitati iz vrednosti *excessFlow* promenljive čvora koji je ponor. Na slikama 5 i 6 može se videti način vizualizacije push operacije i dobijanja krajnjeg rezultata.

Iako je ovaj algoritam efikasniji od Edmonds-Karpovog, postoje načini da se i njegova efikasnost unapredi. Glavnu prepreku u ostvarivanju efikasnosti predstavlja ponovni obilazak svih čvorova u svakoj iteraciji prilikom potrage za aktivnim čvorom. Ispostavlja se da nije potrebno uvek obilaziti



Slika 5: Grafički prikaz izvršavanja jedne push operacije



Slika 6: Grafički prikaz rezultata izvršavanja algoritma

sve čvorove, već je moguće čuvati listu aktivnih čvorova i nju ažurirati. Pristup listi vrši se u amortizovanom vremenu O(1), što doprinosi smanjenom vremenu izvršavanja algoritma. Na samom početku, lista sadrži samo susede izvornog čvora, dok se kasnije, nakon izvršene *push* operacije u nju dodaju čvorovi koji su primili deo toka. Kada se iz aktivnog čvora više ne može propustiti tok ni u jedan od njegovih suseda, potrebno ga je ponovo obeležiti. Kada u njemu nema više viška toka, on se briše iz liste aktivnih. Složenost ovako optimizovanog algoritma je $O(V^3)$, jer se aktivan čvor biti izabran O(V) puta, a kompleksnost *relabel* operacije je $O(V^2)$.

Takođe treba napomenuti da postoje i različiti načini da se odabere aktivni čvor. Pametnim izborom aktivnog čvora može se dodatno smanjiti vremenska kompleksnost algoritma. Postoje sledeći načini selekcije:

- *FIFO selekciono pravilo* podrazumeva da se aktivni čvorovi čuvaju u redu. Nakon svakog propuštanja toka element sa početka reda se uklanja, dok se u red dodaje čvor koji je sada postao aktivan. Složenost ovog algoritma je $O(V^3)$.
- *Obeleženi na početak selekciono pravilo* podrazumeva da se čvor koji je ponovo obeležen *relabel* operacijom bude pomeren na početak liste aktivnih čvorova, te da se iz njega dalje propušta tok. Složenost ovog algoritma je takođe $O(V^3)$.
- *Selekciono pravilo najveće oznake* podrazumeva da se u svakom koraku od svih aktivnih čvorova bira onaj koji ima najveću visinu. Složenost ovog algoritma je takođe $O(V^2\sqrt{E})$.

Za potrebe ovog rada implementirani su generički push-relabel algoritam, FIFO i obeleženi na početak selekciono pravilo.

4 Eksperimenti

Nakon upoznavanja sa algoritmima, ovo poglavlje biće posvećeno testiranju programa i analizi dobijenih rezultata. Najpre su algoritmi pokrenuti nad transportnim mrežama koje su dostupne na internetu radi provere njihove valjanosti. Radi lakšeg snalaženja, ovi test primeri nazvani su NodesNEdgesM.json, gde su N i M redom broj čvorova i grana od kojih se test primer sastoji. Svi algoritmi dobro su se pokazali, tj. davali su korektne rezultate. No, kao što se može videti iz naziva fajlova, svi pronađeni test primeri sastoje od malog broja čvorova i grana, te je za detaljniju analizu potrebno je generisati masivnije grafove.

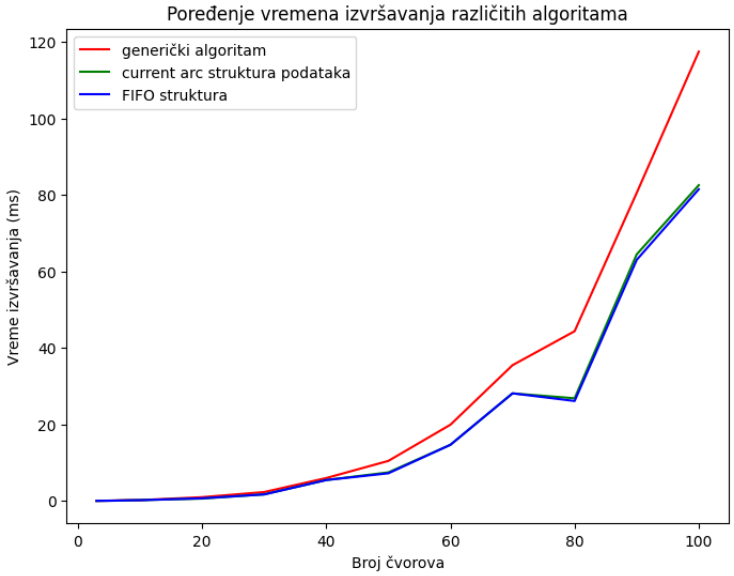
Za postizanje ovog cilja, napisan je program generateTests.ipynb čija je uloga da generiše usmerene grafove sa željenim brojem čvorova i nasumično odabranim granama. Zatim je potrebno od datog grafa napraviti transportnu mrežu procesom koji je opisan u uvodnoj sekciji. Na kraju, potrebno je generisane grafove smestiti u odgovarajuće .json fajlove kako bi ih bilo lakše učitati i testirati algoritme na njima. Generisani su grafovi sa 20, 30, 40, 50, 60, 70, 80, 90 i 100 čvorova, a ovi test primeri nazvani su NodesN.json, gde N označava broj čvorova od kojih se test primer sastoji.

Rezultati izvršavanja algoritama mogu se naći u tabeli 4, a poređenje vremena izvršavanja na slici 7. Kao što se iz priloženog može videti, current-arc i FIFO strukture i u praksi su se pokazale dosta bolje od generičkog algoritma, pogotovo za veće ulazne vrednosti. Kako su oba efikasna algoritma iste složenosti, ne treba da čudi što su i vremena izvršavanja prilično ujednačena, s tim što FIFO struktura ipak pokazuje za nijansu bolje performanse u svim navedenim primerima.

Tabela 1: Pregled vremena izvršavanja algoritama

V	Generički (ms)	Current-arc (ms)	FIFO (ms)	Gustina
3	0.0496	0.0251	0.0210	0.33
4	0.05687	0.054	0.0469	0.42
6	0.1123	0.092	0.0889	0.3
8	0.1995	0.152	0.163	0.27
10	0.2255	0.222	0.2029	0.17
20	0.9851	0.624	0.696	0.14
30	2.328	1.723	1.712	0.12
40	5.9887	5.474	5.464	0.11
50	10.466	7.466	7.226	0.11
60	19.9287	14.686	14.67	0.09
70	35.4815	28.11	28.12	0.10
80	44.3591	26.83	26.59	0.10
90	80.507	64.44	63.01	0.10
100	117.419	82.525	81.47	0.10

Pored veličine ulaza, bilo bi zanimljivo proveriti da li još neki atribut grafa utiče na vreme izvršavanja algoritama. Stoga je za svaki test primer izračunata i *gustina grafa* – odnos broja grana koje se nalaze u grafu i ukupnog mogućeg broja grana. Međutim, odgovor na ovo pitanje je negativan, tj. rezultati izvršavanja ne zavise direktno od gustine grafa što se može videti iz priložene tabele.



Slika 7: Poređenje vremena izvršavanja algoritama

5 Zaključak

Kao što je u radu i izloženo, problem nalaženja maksimalnog toka kroz transportnu mrežu može biti veoma značajan u svakodnevnom životu. Algoritmi koji su bili izloženi pokazali su se dobro u poređenju sa jednostavnim Edmonds-Karpovim algoritmom. Takođe, za razliku od Ford-Fulkersonovog algoritma, svaki od navedenih algoritama se garantovano završava nezavisno od ulaznih vrednosti. Očekivano, testovi su pokazali da napredne strukture kao što su current-arc i FIFO postižu dosta bolje rezultate od generičkog algoritma za sve moguće ulaze. Treba napomenuti i da vreme izvršavanja algoritama zavisi direktno od broja čvorova koji se nalaze u grafu. Korelacija između gustine grafa i vremena izvršavanja algoritama nije pronađena. Najzad, zbog značaja transportnih mreža potrebno je nalaziti nove mogućnosti za napredak u ovom polju – bilo nalaženjem novih algoritama bilo optimizacijom postojećih.

Literatura

[1] Andrew Goldberg, Eva Tardos, and Robert Tarjan. *Network Flow Algorithms*. Princeton University, Department of Computer Science, 1990.

[2] Vesna Marinković, Filip Marić, Strahinja Stanojević, and Sana Stojanović-Đurđević. *Konstrukcija i analiza algoritama*. Matematički fakultet, Univerzitet u Beogradu, 2019.

[3] Sargam Monga. Overview of Maximum Flow Problem. on-line at: https://iq.opengenus.org/maximum-flow-problem-overview/.

[4] Ferozuddin Riaz and Khidir Ali. Applications of graph theory in computer science. *Third International Conference on Computational Intelligence, Communication Systems and Networks*, 2011.

[5] Univerzitet u Kaliforniji. Applications of Network Flow. online at: http://www.cs.ucf.edu/~dmarino/progcontests/modules/netflow2/NetFlow-Apps.pdf.